(NeXT Tip #44) Compiling for NeXTSTEP 486

Christopher Lane (*lane[at]CAMIS.Stanford.EDU*)
Thu, 28 Oct 1993 14:54:18 -0700 (PDT)

To: KSL-NeXT[at]CAMIS.Stanford.EDU
Message-ID: <MailManager.751845258.7509.lane[at]ssrg-next-1>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII

Just some simple (and probably obvious) guidelines about recompiling NeXTSTEP applications for use on the NeXTSTEP 486 machines:

You should really try to start with an application that's been rebuilt and debugged for NeXTSTEP 3.0 or later. You can generally tell if this is the case if the application source directory has a PB.project file -- that is, the application components are under the control of ProjectBuilder. Applications prior to 3.0 use the project management portion of InterfaceBuilder and will typically have an IB.proj file. However just because an application has an IB.proj file that doesn't mean there isn't a PB.project file as well -- although obsolete under 3.x, the IB.proj files don't necessarily get deleted in the upgrade process.

There are a couple ways to compile under NeXTSTEP 3.1, thin (single architecture) or MAB (multiple architecture aka fat). An application complied MAB will run on either system and is what you probably want to generate. Compiling an application 'thin' doesn't mean that it won't run on a 486 machine, a non-MAB (thin) compile defaults to the type of machine you happen to be compiling on, either m68k or i486. You can generate MAB binaries, or thin binaries for either architecture from either kind of NeXT, the compiler under 3.1 is a full cross-compiler on both systems.

The obvious exception to the 'compile MAB' rule is during development as, no surprise, compiling MAB takes twice as long as compiling thin. You should probably compile MAB anything you make available to others. (Also, no surprise, a MAB binary is slightly more than twice as large as a thin one.)

If you just do 'make' in your 3.0 application directory, you will end up with a thin executable for the type of machine you compiled on. Although it is possible to pass arguments to 'make' to create a MAB application:

```
make RC_ARCHS='m68k i486' RC_CFLAGS='-arch m68k -arch i486'
```

I strongly recommend using ProjectBuilder instead to rebuild your application. ProjectBuilder has an 'Options...' panel in the Builder section that will let you check off which architectures to build for. Also, ProjectBuilder provides a nice interface for getting to sources of problems that the compiler detects.

Of course, both approaches only work with ProjectBuilder generated 'make' files. If you're working with a generic 'C' package, you'll have to modify the Makefile to include the '-arch m68k -arch i486' options to 'cc' directly.

Most 3.0 applications will compile cleanly for the 486 architecture, but not all. Some will fail on subtle differences between 3.0 & 3.1 but these should fail compilation for either system. (E.g., header files that have moved.)

Some may fail on missing libraries that we imported independent of the official NeXT release. A typical example of this libtext.a, a useful library dropped by NeXT that we've been bringing forward, as a binary, from release to release. Some other libraries, like libcs.a, I've located the current sources for and have recompiled MAB so it can still be used.

Just because your application compiles cleanly on the 486 system, that doesn't guarantee that it will run correctly, unfortunately. A classic example of this problem is the big and little 'endian' issue. The byte order of various types is reversed between the m68k and i486 architectures and this can be a problem when reading in files of saved or 'standard format' data. For the most part, if you just work with NeXT's typed streams, you'll never need to know this but even when using typed streams the bit field construct in the 'C' language can get you into serious trouble.

The document /NextLibrary/Documentation/NextDev/Concepts/PortabilityGuide.rtf discusses this in detail and explains one compatible way to code bit fields.

If you have a working 2.0 application, with sources, that compiles and runs fine under 3.x as is and you don't want to go to the trouble to update it to a 3.x-style application -- you can still generate a MAB executable for use on either system. The trick is to do a 'split compile' by doing a command line 'make' on one system, do it again on the other architecture and then, within the *.app directory, combine the individual thin executables into a single MAB binary using 'lipo'. (See the lipo man page for details.) This gets messy when working with applications with multiple executable components. I don't recommend this approach, but sometimes it's the path of least resistance.

(This trick can also work for compiling generic C programs into MAB binaries.)

Compiling an application MAB under NeXTSTEP 3.1 doesn't necessary make it incompatible with NeXTSTEP 3.0 -- though it will be incompatible with NeXTSTEP 2.x and earlier. There is some forward compatibility built into 3.0 to support MAB applications, but it doesn't always work. Typical problems are MAB 'bundles' which 3.0 doesn't handle correctly as well as the some basic differences between 3.0 & 3.1 system libraries. It's nice if you can take advantage of this forward compatibility feature, but don't rely on it.

- Christopher